Universidad Zaragoza

1542

Trabajo fin de grado

Binarización de descriptores en redes neuronales profundas
Binarization for deep neural networks features

Mikel

Mikel Martínez Iparraguirre

Director

Javier Civera Sancho

ESCUELA DE INGENIERÍA Y ARQUITECTURA
2018

TRABAJOS DE FIN DE GRADO / FIN DE MÁSTER

Escuela de Ingeniería y Arquitectura Universidad Zaragoza

DECLARACIÓN DE AUTORÍA Y ORIGINALIDAD

(Este documento debe acompañar al Trabajo Fin de Grado (TFG)/Trabajo Fin de Máster (TFM) cuando sea depositado para su evaluación).

D./Dª. Mikel Martínez Iparraguirre,

con nº de DNI 73159207W en aplicación de lo dispuesto en el art. 14 (Derechos de autor) del Acuerdo de 11 de septiembre de 2014, del Consejo de Gobierno, por el que se aprueba el Reglamento de los TFG y TFM de la Universidad de Zaragoza,

Declaro que el presente Trabajo de Fin de (Grado/Máster) Grado, (Título del Trabajo) Binarización de descriptores en redes neuronales profundas

es de mi autoría y es original, no habiéndose utilizado fuente sin ser citada debidamente.

Zaragoza, 21 de septiembre de 2018

Fdo:

AGRADECIMIENTOS

Me gustaría agradecer a Javier Civera y a José María Fácil su paciencia, su tiempo, sus explicaciones y todos los conocimientos que han sido capaces de transmitirme a lo largo del proyecto. Un proyecto que no podría haber llevado acabo sin la tutela de dos expertos como lo son ellos. A mi familia por haberme ayudado directa o indirectamente a tomar decisiones lo lago de todos estos años y en especial a mi padre, por ser esa inagotable fuente de conocimiento. A Ana, por haber estado ahí desde el principio, sufriéndome y aguantándome con una sonrisa.

RESUMEN

La binarización de descriptores en redes neuronales profundas es un área que se encuentra en investigación perteneciente al aprendizaje automático. Uno de los retos a resolver en redes neuronales es reducir el coste computacional y temporal en la clasificación de imágenes. Por ello se propone trabajar en el espacio binario en vez de en el real. La binarización de descriptores consiste en, transformar un descriptor extraído por la red compuesto por números reales en uno compuestos por bits. Para que los descriptores binarios mantengan la información sintetizada en el descriptor real, se proponen unas arquitecturas de red en las que se tiene un control sobre la localización de los descriptores en el espacio.

La implementación del método de binarización conlleva varios desafíos. Por un lado, disponer de datos adecuados para el entrenamiento de la red y para la comprobación de los resultados, para lo que se ha creado un conjunto de datos adecuado a nuestra red a partir del MNIST. Por otro lado, la elección de la estructura, el proceso de entrenamiento de las redes, así como el diseño e la implementación de nuevas funciones de coste. Tras evaluar exhaustivamente las diferentes arquitecturas para el MNIST, se ha comprobado que el método de binarización propuesto DCCB, alcanza con una precisión del 96,6 % el segundo puesto en el estado del arte en la binarización de descriptores para el conjunto de datos del MNIST.

Índice

Capítulo 1

Introducción y objetivos

El trabajo pertenece al área de la visión por computador, rama de la ciencia que estudia, analiza e interpreta las imágenes mediante sistemas computacionales, transformando el mundo real que percibimos con nuestros ojos en una representación numérica y simbólica para computadores. Problemas como el reconocimiento facial y de objetos, (ver Figura 1.1), o de localización y mapeo simultáneos (SLAM), son objeto de la visión por computador. En este trabajo vamos a centrarnos en el estudio de las redes neuronales profundas para aplicaciones de reconocimiento.

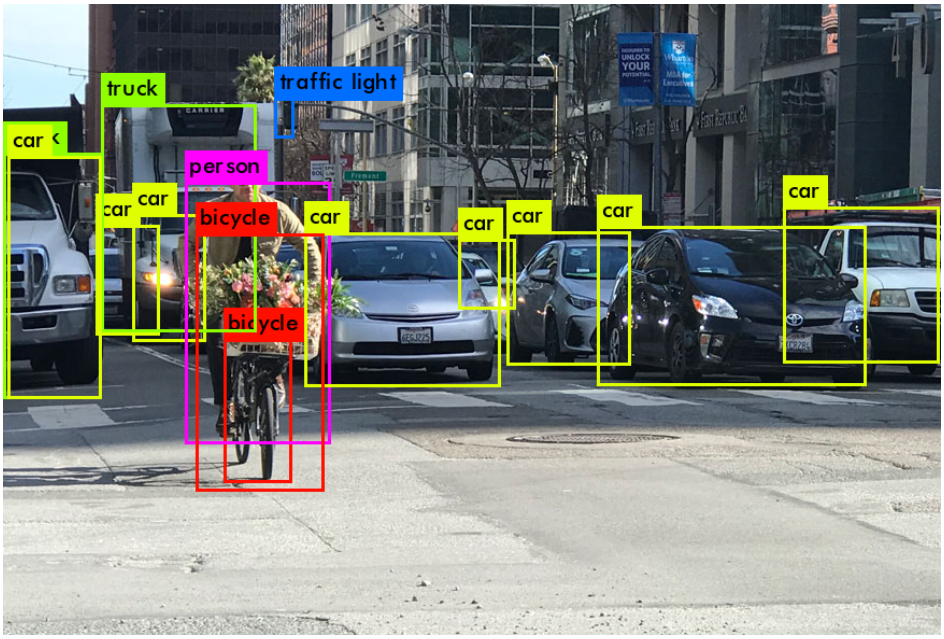


Figura 1.1: Ejemplo del reconocedor de objetos YOLO.

Las redes neuronales tienen la capacidad de reconocer patrones en nueva imagen de entrada similar a aquellas con las que la red ha sido entrenada. Primero se realiza un entrenamiento con grandes bases de datos de las que la red aprende los patrones más representativos para aprender a clasificar categorías. Una red sintetiza la información de los datos de entrada en descriptores, vectores de números reales, reduciendo la

dimensión y aumentando la invarianza dentro de una misma categoría.

El emparejamiento de descriptores se realiza calculando su distancia euclídea, perteneciendo a la misma categoría aquellos cuya distancia sea menor. Uno de los retos a resolver en redes neuronales es reducir el coste computacional en los emparejamientos de descriptores, así como reducir el espacio en memoria que ocupan. Por ello, en este trabajo se propone trabajar con descriptores binarios, vectores de datos tipo *bit* en vez de tipo *float* o *double*. El emparejamiento de descriptores binarios se realiza calculando la distancia de Hamming, perteneciendo a la misma categoría aquellos cuya distancia de Hamming sea menor. La distancia de Hamming entre dos vectores binarios de misma longitud es el número bits en el que difieren, (ver Figura 1.2). Al trabajar con distancias de Hamming en vez de distancias euclídeas el coste computacional por emparejamiento puede verse reducido del orden de 35 veces [1].

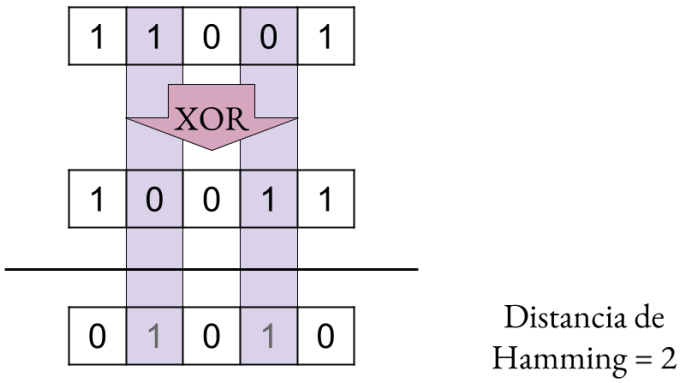


Figura 1.2: Distancia de Hamming entre dos descriptores binarios. La distancia de Hamming equivale a realizar la operación lógica XOR y a sumar los elementos del vector resultante. Por cada bit distinto la distancia de Hamming se incrementa una unidad.

Sin embargo, al trabajar con descriptores binarios reducimos el dominio de representación, ya que pasamos de un espacio de representación continuo a uno discreto con dos valores. Así pues, el máximo número de clases que se pueden representar con un descriptor binario de n dimensiones es 2^n.
El objetivo del trabajo es proponer un método para que las redes generen descriptores continuos fáciles de binarizar mediante la aplicación de una función de umbralización, (ver Figura 1.3). Se parte de una arquitectura de red neuronal siamesa a la que se le han añadido diferentes arquitecturas y funciones de coste diseñadas e implementadas

en Caffe [2], para alcanzar el objetivo planteado.

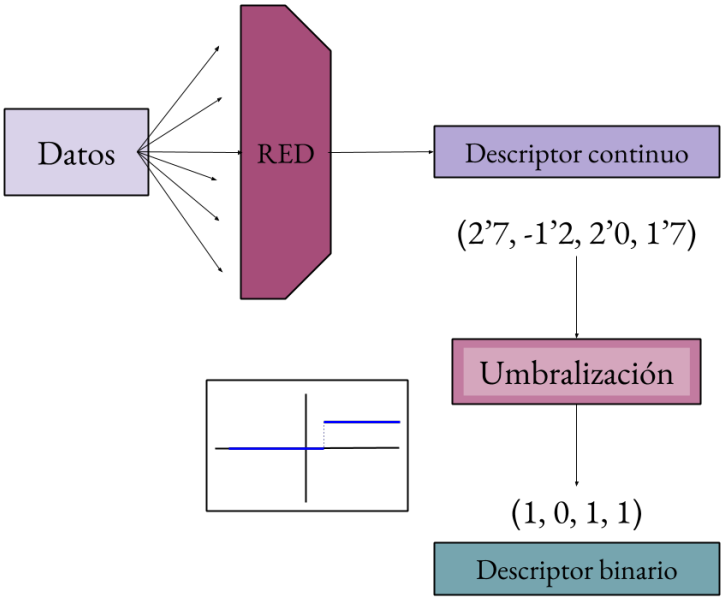


Figura 1.3: Esquema del proceso de binarización. A la salida de la red aplicamos la función de umbralización a cada coordenada del descriptor obteniendo así su relativo binario.

En los capítulos 1 y 2 se explican conceptos básicos de redes neuronales así como las bases de datos utilizadas. Posteriormente, en el capítulo 3 se detalla la binarización en alto nivel y en el 4 las funciones diseñas para el desarrollo del método. Finalmente en los capítulos 5 y 6 se muestran los resultados de los experimentos y las conclusiones respectivamente.

Capítulo 2

Redes neuronales profundas

En este apartado se explican una serie de conceptos básicos necesarios para la comprensión del resto del trabajo.

Las redes neuronales profundas son un modelo perteneciente al ámbito del aprendizaje automático. Esta disciplina tiene su origen en la década de 1950. Sin embargo tenía dos grandes limitaciones: la pequeña potencia y memoria de los computadores de la época y la falta de grandes conjuntos de datos para el entrenamiento. Por ello, no es hasta finales de los 90 que gracias al desarrollo de la capacidad de computación, de memoria y de datos disponibles el *"Deep learning"* resurge con resultados mejores que cualquier otra técnica en gran variedad de problemas.

Las redes neuronales proporcionan algoritmos capaces de aprender funciones complejas mediante el entrenamiento en conjuntos de datos. Estamos interesados en su capacidad para el reconocimiento de patrones a partir de imágenes que permiten identificar categorías: objetos, rostros o lugares entre otros.

2.1. Neurona artificial

La neurona artificial es la unidad básica de las redes neuronales. Una neurona procesa por si misma un vector n-dimensional de entrada $\mathbf{x} = (x_1, x_2, ..., x_n)^\top$, para generar una salida y. Cada neurona tiene una serie de pesos asociados $\mathbf{w} = (w_1, w_2, ..., w_n)^\top$. La neurona realiza el producto escalar entre la entrada $\mathbf{x}$ y los pesos $\mathbf{w}$ y le suma un sesgo b (en inglés *bias*). Posteriormente se aplica una función de activación no lineal, $f : \mathbb{R} \to \mathbb{R}$, (ver Figura. 2.1).

$$y = h_{\mathbf{w},b}(\mathbf{x}) = f\left(b + \sum_{i=1}^{N} w_i x_i\right) \tag{2.1}$$

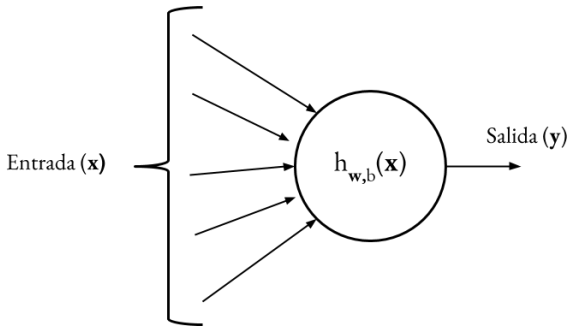


Figura 2.1: Esquema neurona aritificial

2.2. Redes neuronales profundas

Una red neuronal es un conjunto de neuronas artificiales distribuidas y agrupadas en capas interconectadas. Existen tres tipos de capas atendiendo a su localización en la red (Figura. 2.2). La *Capa de entrada* es aquella por la que se introducen los datos (imágenes en nuestro caso), con tantas neuronas como dimensiones tenga los datos de entrada. La *Capa de salida* es la que proporciona la salida deseada, en nuestro caso el descriptor de salida a partir del cual se clasifica la imagen. Las *Capas ocultas* son toda capa situada entre la capa de entrada y de salida, puede estar precedida por otra capa oculta o por la capa de entrada.

En una red neuronal la entrada $\mathbf{x}$ proporciona la información inicial que se propaga a través de las capas. La propagación va desde la capa de entrada hasta la capa de salida para finalmente obtener el descriptor. Este método se denomina *propagación hacia delante* o *forward-propagation*. Durante el entrenamiento el *forward-propagation* se ejecuta para obtener una función de coste que modela la discrepancia entre la salida de la red y la salida deseada. Es durante el entrenamiento, cuando la red reajusta sus hiper-parámetros con el fin de minimizar dicha función de coste. Para corregir este error se emplea el método de aprendizaje stochastic gradient descent (SGD), método de aprendizaje que minimiza la función de coste propagando hacia atrás los gradientes con el fin de reajustar los pesos de la red. Este método trabaja con una muestra de la base de datos denominada *batch*, ya que procesar toda la base de datos en cada iteración supondría un coste computacional demasiado alto. El método no garantiza que el aprendizaje converja al mínimo global, ya que puede quedarse atrapado en mínimos

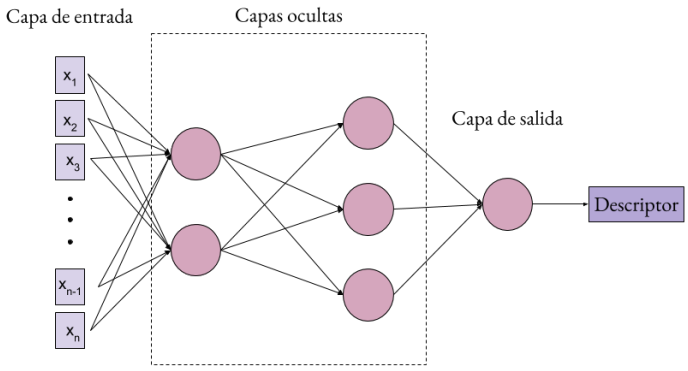
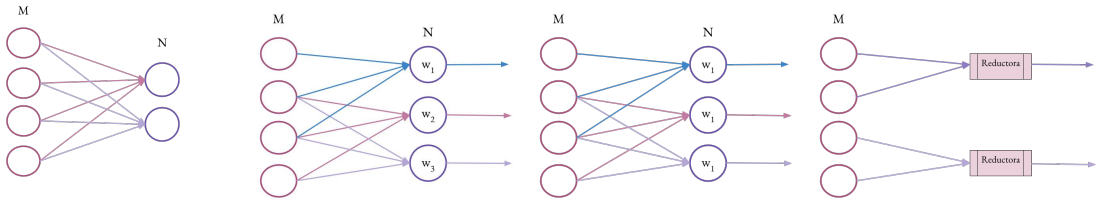


Figura 2.2: Esquema de una red neuronal. Capa de entrada, ocultas, de salida y el descriptor.

locales. Sin embargo, afortunadamente este algoritmo logra que la red aprenda.

La *propagación hacia atrás* o *back-propagation* es el método de cálculo de derivadas utilizado en *deep learning*. Para que el entrenamiento converja, los pesos han de incializarse a valores muy próximos a cero. En la práctica dicha inicialización se realiza de forma aleatoria.

A continuación se explican las estructuras de capas utilizadas en el trabajo.



(a) (b) (c) (d)

Figura 2.3: Varios tipos de capas de una red neuronal. M es la capa anterior y N = M+1 la actual. (a) Capa totalmente conectada. (b) Capa localmente conectada. (c) Capa convolucional, tiene todos los pesos compartidos. (c) Capa reductora.

Función de coste *(Loss)*: es la función que modela el error cometido en el entrenamiento. Esta capa está localizada al final de la red y devuelve un escalar entre cero e infinito que permite posteriormente reajustar los pesos. Desde este punto de vista el entrenamiento es un problema de optimización en el que se trata de encontrar la configuración de pesos que minimicen la función de costes. Es por ello, que la elección

de la función de coste sea tan determinante para el entrenamiento. Ejemplos de capas de coste son las denominadas contrastive, la softmax o la euclideana.

Capa totalmente conectada *(fully-connected)*: En esta capa, cada neurona se conecta a todas las de la capa anterior. Esto permite que pueda tener en cuenta la entrada al completo en su procesamiento, con la desventaja de tener un elevado número de parámetros. Esto se debe a que cada neurona artificial tiene tantos pesos como entradas, junto a un sesgo que deben aprender y almacenar.

Capa localmente conectada : Es un tipo de capas en las que sus neuronas se conectan solo a un conjunto de la capa anterior. La finalidad es distinguir patrones locales dentro de las imágenes de entrada con un coste de entrenamiento menor ya que el número de parámetros utilizado es menor que en las totalmente conectadas.

Capa reductora *(pooling)*: Estas capas no tienen parámetros que aprender, aplican una función conocida sobre la entrada. Una de esas funciones es la "max pooling", que funciona dejando pasar los valores máximos de una entrada y descartando el resto. Al reducir el tamaño de los vectores con los que trabaja la red, se disminuye el número de conexiones y por tanto de parámetros que se necesitan. En el caso de trabajar con imágenes también se aumenta la robustez ante la traslación local.

Capa convolucional: es un tipo de capa localmente conectada. Tienen una conexión especial que hace que implementen la operación matemática de convolución. Las neuronas de estas capas se conectan al mismo número de neuronas de la capa anterior y todas comparten los mismos pesos. Estas capas presentan invarianza espacial, es decir, si una capa convolucional se especializa en detectar ruedas en imágenes, la capa podría detectar las ruedas en cualquier parte de la imagen. Las capas total y localmente conectadas también pueden llegar a alcanzar la invarianza espacial mediante el entrenamiento, si bien es mucho más complicado ya que requieren más tiempo y más datos para ello. Las convolucionales lo consiguen directamente, gracias a su estructura y sus conexiones, utilizando también menos parámetros.

2.3. Red siamesa

Una red siamesa es un modelo de entrenamiento que consiste en dos redes idénticas dispuestas en paralelo, que en el entrenamiento comparten todos sus pesos. Este tipo de red consta de dos entradas, a y b, una para cada rama. Si la imagen de la entrada a fuese la misma que la de la entrada b, la red proporcionaría el mismo descriptor tanto a la salida de a como a la de b. En nuestra red siamesa utilizamos como función de coste denominada contrastive que se explicará detalladamente en el apartado 2.3.1. En el ejemplo de la Figura 2.4 podemos ver como se comporta una red siamesa ante las dos

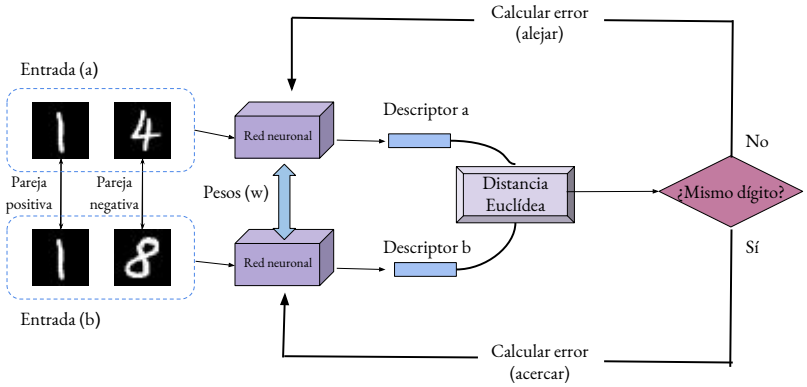


Figura 2.4: Diagrama de una red siamesa con dos ejemplos, uno positivo y otro negativo

situaciones posibles, una entrada de pareja positiva o de pareja negativa. Cada pareja positiva utiliza dos imágenes diferentes con el mismo dígito, y cada pareja negativa dos imágenes diferentes con distintos dígitos. Tras ser procesadas por la red, se obtiene el descriptor a y el descriptor b. En el caso de ser pareja positiva, la distancia euclídea entre el descriptor a y el b se tenderá a reducir. En el caso de ser pareja negativa, la distancia euclídea tenderá a aumentar. Por ello, una red siamesa acerca aquellas imágenes con mismos dígitos, y aleja aquellas en las que son diferentes. Esta topología de red no requiere del conocimiento del número de clases, dado que la red solo distingue entre casos positivos y negativos. En la Figura 2.5 se puede ver la arquitectura de la red siamesa empleada a lo largo del trabajo. El recuadro del final de la red es la zona donde van a desarrollarse las diferentes funciones utilizadas para la binarización de descriptores.

2.3.1. Función de coste contrastive

La función de coste contrastive [3] acerca los descriptores cuando su etiqueta es 1, pareja positiva, y los aleja cuando su etiqueta es 0, pareja negativa.

$$L = yd^2 + (1 - y)\,\text{máx}(m - d, 0)^2 \tag{2.2}$$

Donde L es el coste, d es la distancia euclídea entre los descriptores de la pareja e y es la etiqueta de la pareja y m es el margen. La función contrastive separa y acerca

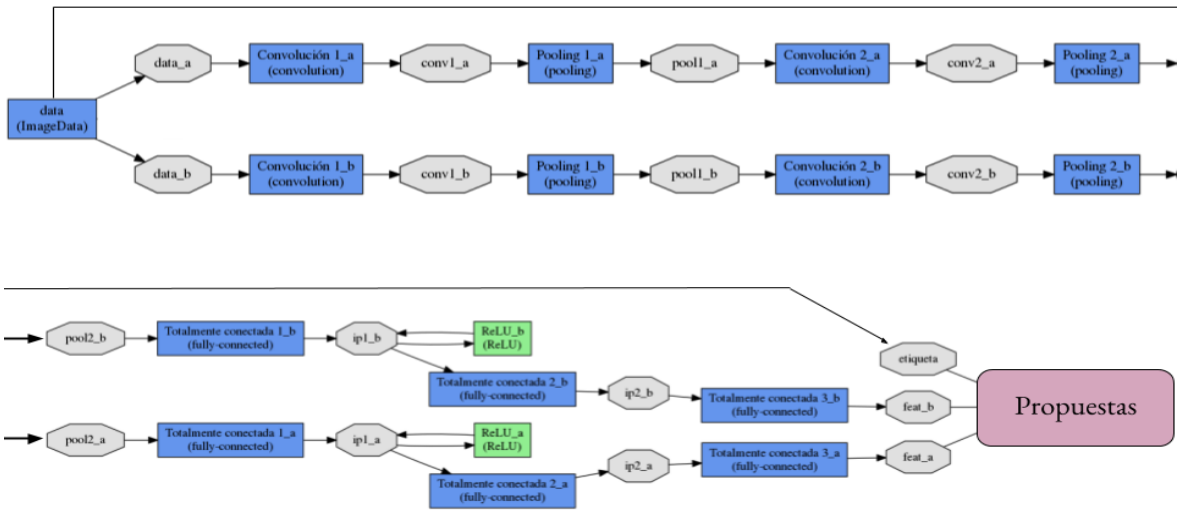


Figura 2.5: Topología de la red siamesa a entrenar. El recuadro morado es la parte de la red en la que se localizan las distintas arquitecturas propuestas.

los descriptores de forma relativa. Sin embargo, si para una pareja negativa d es mayor que m, que usualmente toma el valor 1, el coste de la función es cero y no aumenta su distancia. Debido a esta saturación los clusters quedan en torno a los valores de inicialización. La función de coste contrastive permite agrupar en clusters separables las clases durante el entrenamiento. En la Figura 2.6 se puede ver la agrupación resultante del entrenamiento de una red siamesa con una función contrastive.

2.4. Funciones de activación típicas

Para que la red aprenda patrones complejos se deben utilizar funciones de activación no lineales $f : \mathbb{R} \rightarrow \mathbb{R}$. De no ser así, el aprendizaje sería una combinación lineal de todas las funciones de activación. Las funciones más típicas son la función sigmoide y la función ReLU.

2.4.1. Función sigmoide

La función sigmoide:

$$f(x) = \frac{e^x}{e^x + 1} \tag{2.3}$$

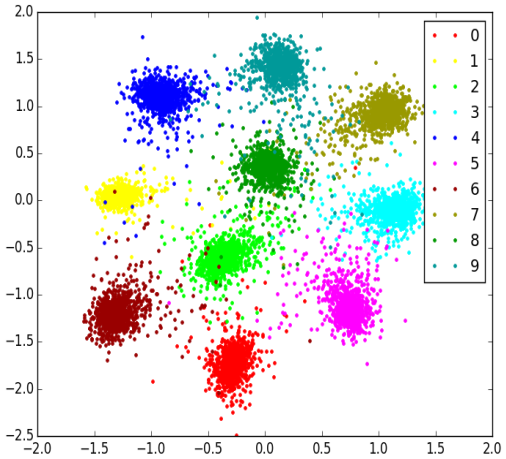


Figura 2.6: Agrupación con descritores 2D para el problema del MNIST etrenado con red siamesa con contrastive loss. Nótese la correcta separación de las diez clases.

es una función de activación similar a la función de binarización (eq. 4.1). Esta función es continua y derivable en todo el dominio de los reales y su recorrido es el intervalo [0,1] (ver Figura. 2.7). La función sigmoide tiene una región con fuerte derivada entorno al cero, sin embargo fuera de esa región la derivada es casi nula. Es por ello, que la función sigmoide satura a 0 los valores negativos y a 1 los positivos. Debido a la naturaleza de la función, en la back-propagation el resultado de la función de costes estará multiplicado como máximo por un cuarto de su valor, ya que el recorrido de la derivada es [0,0.25]. Cuanto más profunda sea la red, la tasa de aprendizaje será menor debido a que el valor del gradiente se ve más reducido.

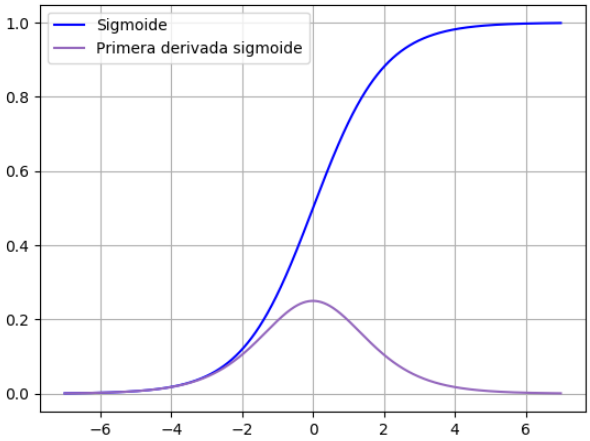


Figura 2.7: Función sigmoide en azul y su derivada en morado.

2.4.2. Rectified Linear Unit

La Rectified Linear Unit, denominada ReLU, es la función de activación no lineal más simple que podemos encontrar. Esta función satura a cero los valores negativos y mantiene su valor a los positivos, (ver Figura 2.8). Su derivada es 1 para valores mayores que cero, por lo que durante la *back-propagation* no tiene un efecto reductor en el valor del coste. Esta función es continua en el dominio de los reales y tiene como recorrido el intervalo $[0, \infty)$.

$$f(x) = \text{máx}(0, x) \tag{2.4}$$

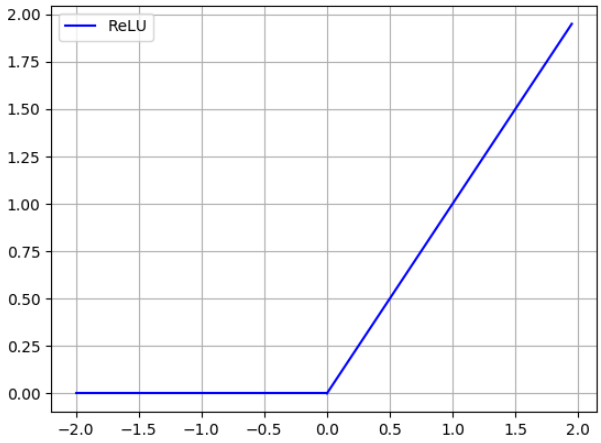


Figura 2.8: Gráfica de la función ReLU

Capítulo 3

Bases de datos

MNIST es una base de datos de dígitos manuscritos de diez clases, los números del 0 y al 9, (ver figura 3.1) frecuentemente utilizada como banco de pruebas en redes neuronales. La base de datos consta de 60.000 imágenes para el entrenamiento y 10.000 para el testeo, cada imagen con 28x28 píxeles en escala de grises de 8 bits. Se han estructurado dos bases de datos, dependiendo de si la arquitectura de la red tenía la función de coste función de coste distribución homogénea.

- Arquitectura sin la función de coste distribución homogénea. En esta base de datos las etiquetas son 1 o 0, en caso de ser una pareja positiva o negativa respectivamente. Las parejas positivas son dos imágenes diferentes con el mismo dígito y las parejas negativas son dos imágenes con distinto dígito. Para los diferentes experimentos se generarán otras tres bases de datos dependiendo del número de clases a entrenar, 4, 8 o 10, como puede verse la tabla 3.1.

- Arquitectura con la función de coste distribución homogénea. En esta base de datos las parejas son todas positivas. La base de datos tendrá que estar ordenada de tal forma que cada batch, que será de igual tamaño que el número de clase a entrenar, no tenga ninguna clase repetida. En la tabla 3.2 se puede ver un ejemplo para entrenar cuatro clases.

Entrada a	**Entrada b**	**Etiqueta**
1	*1*	1
1	*0*	0
2	*5*	0
4	*4*	1
...	...	...

Tabla 3.1: Ejemplo de una base de datos siamesa para 10 clases. En la primera columna las imágenes con los dígitos que van a entrar en la rama (a) de la red siamesa. En la segunda columna las imágenes con los dígitos que van a entrar en la rama (b) de la red siamesa. En la tercera columna las etiquetas.

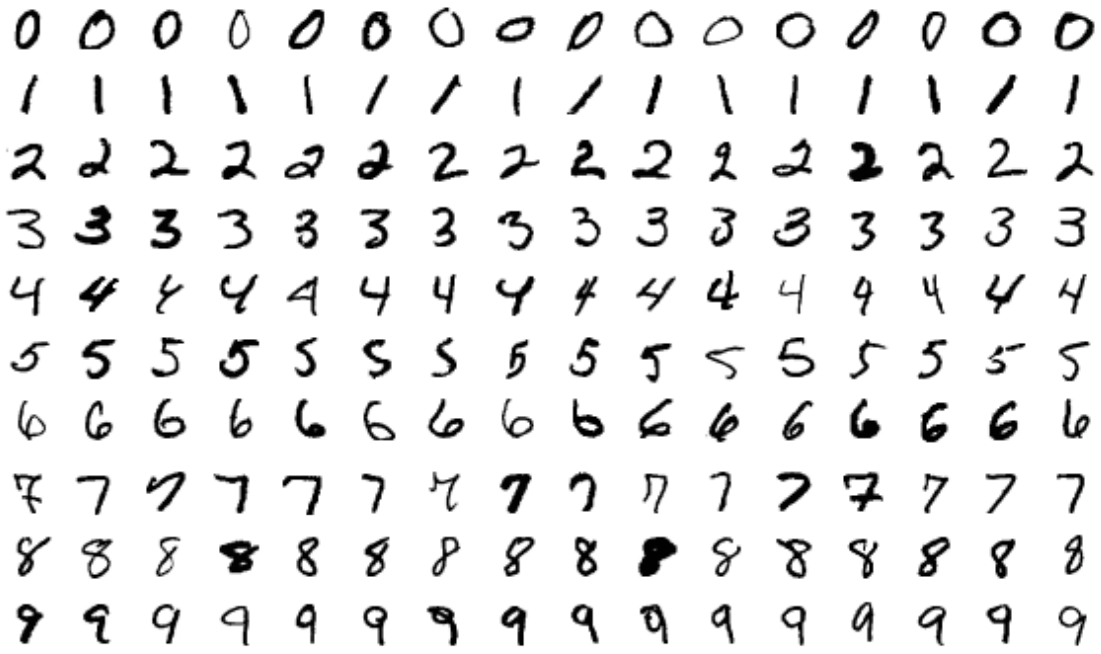


Figura 3.1: Ejemplo imágenes del MNIST. Las imágenes de cada fila pertenecen a la misma clase.

Entrada a	**Entrada b**	**Etiqueta**
0	*0*	*1*
1	*1*	*1*
2	*2*	*1*
3	*3*	*1*
0	*0*	*1*
1	*1*	*1*
...	...	*1*

Tabla 3.2: Estructura base de datos para entrenamiento de red siamesa con función de coste distribución homogénea para 4 clases.

Capítulo 4

Binarización de descriptores

La binarización propuesta consiste en obtener un descriptor binario a partir de un descriptor continuo, aplicando la binarización trivial (eq.4.1) a todas las coordenadas del descriptor. Para que la binarización sea eficiente los descriptores continuos han de estar agrupados por clases en clusters y localizados en el espacio acorde con el umbral de binarización, en nuestro caso 0.5, (ver figura 4.1). Por ello el entrenamiento se basa en agrupamiento de descriptores y umbralización de los clusters. Para que la umbralización sea correcta se ha de intervenir la localización de los clusters en el espacio continuo. Las funciones de coste tradicionales para agrupamiento no influyen sobre la localización de los clusters y éstos varían de un entrenamiento a otro debido a que la inicialización de los pesos de la red es aleatoria. Es por ello, que en este trabajo se proponen arquitecturas de red que durante el entrenamiento agrupan, localizan y umbralizan los descriptores, forzando a que éstos se distribuyan en torno a las coordenadas cero y uno ocupando el mayor número de bits posible con el fin de maximizar la distancia de Hamming. De este modo, la binarización trivial (eq. 4.1) permite obtener eficientes descriptores binarios.

$$y = 0{,}5\,(\text{sign}\,(x - 0{,}5) + 1) \tag{4.1}$$

Figura 4.1: Función de umbralización en 0.5, correspondiente a la ecuación 3.1.

En los apartados siguientes se explican las funciones de coste diseñadas e implementadas en el trabajo.

4.1. Cuantización binaria

En el estado del arte en la binarización de descriptores con entrenamiento no supervisados encontramos el método propuesto por Kevin Lin en [4], donde propone una serie de funciones que binarizan los descriptores a la salida de una red neuronal invariante ante rotaciones y ruidos. En su artículo el umbral de binarización esta en 0. Las funciones de coste sobre las que se apoya el método son las siguientes:

Función minimal quantization

$$L(W) = \alpha \sum_{n=1}^{N} ||(\mathbf{b}_n - 0{,}5) - F(\mathbf{x}_n, W)||^2 \tag{4.2}$$

Función evenly distributed

$$L(W) = \beta \frac{1}{M} \sum_{m=1}^{M} ||\mu_m - 0{,}5||^2 \tag{4.3}$$

$$\mu_m = \frac{1}{N} \sum_{n=1}^{N} b_n^m \tag{4.4}$$

La función minimal quantization trata de minimizar el error entre el descriptor continuo y su respectivo binario. La evenly distributed trata de ocupar el mayor número de bits posible del descriptor binario.

Se ha propuesto e implementado en C++ nuestra propia versión de (eq. 4.2) que llamamos **función de cuantización** (eq. 4.5) que está centrada en 0.5 y cuya derivada es (eq. 4.6).

$$L(W) = \alpha \sum_{m=1}^{M} \sum_{n=1}^{N} ||b_n^m - F(x_n, W)^m||^2 \tag{4.5}$$

$$L'(W) = -2(b_n^m - F(x_n, W)^m) \tag{4.6}$$

donde L es el coste y L' su gradiente, N es el tamaño del batch y M la longitud del descriptor, $F(x_n, W)^m$ es el valor de la coordenada m del descriptor del elemento n del batch y b_n^m es su valor binario.

La función de cuantización (fig 4.2) es una función continua en todo su dominio, pero no derivable en 0.5. Consta de dos mínimos absolutos en cero y en uno, puntos a los que las coordenadas de los descriptores acabarán acercándose en caso de que el entrenamiento converja.

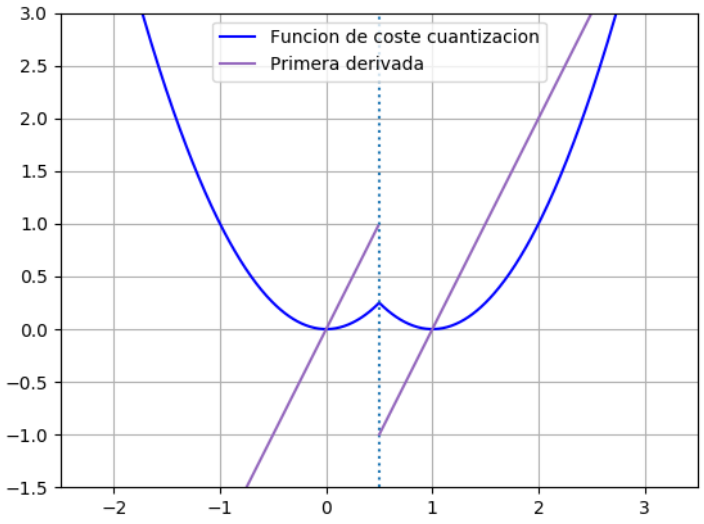


Figura 4.2: Función de coste cuantización (eq. 4.5) en azul. En verde su derivada (eq. 4.6). La línea discontinua marca el umbral de binarización, 0.5.

4.2. Distribución homogénea

Partiendo de la (eq. 4.3) proponemos en este trabajo la función de coste **distribución homogénea** (eq. 4.7) como método para separar los clusters en los cuadrantes binarios, de forma que se maximicen las distancias de Hamming entre las etiquetas binarias de las diferentes clases.

$$L_s(W) = \beta \frac{1}{M} \sum_{m=1}^{M} \left(\|\mu_{m,0} - 0{,}5\|^2 + \|\mu_{m,1} - 0{,}5\|^2 \right) \tag{4.7}$$

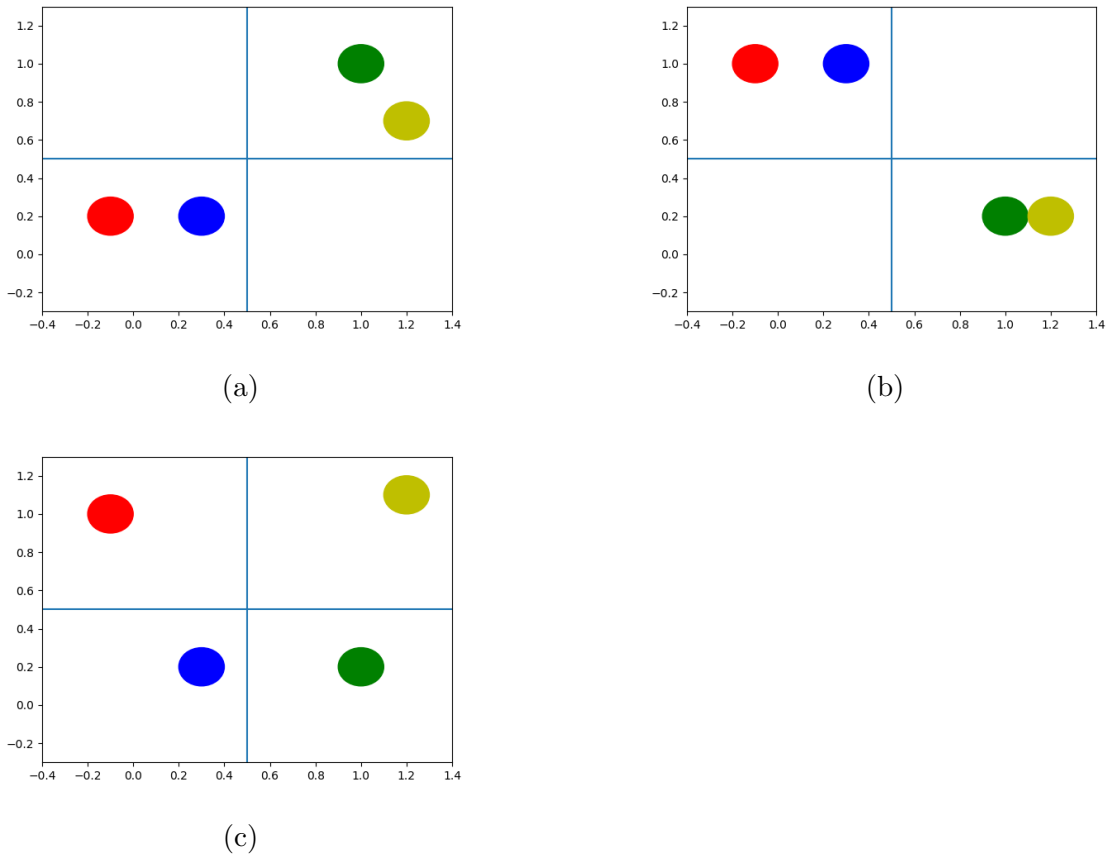
$$\mu_{m,0} = \frac{1}{N_0} \sum_{m \backslash b_n(m-1)=0} b_n^m \tag{4.8}$$

$$\mu_{m,1} = \frac{1}{N_1} \sum_{m \backslash b_n^(m-1)=1} b_n^m \tag{4.9}$$

donde L es el coste de la función, M la dimensión del descriptor, $\mu_{m,0}$ es la media binaria de la coordenada m para los N_0 elementos del batch que tengan la coordenada

Caso	Azul	Roja	Verde	Amarilla	μ_1	μ_2	$\mu_{2,0}$	$\mu_{2,1}$	L_s	L
a	00	00	11	11	0.5	0.5	0	1	0.5	0
b	01	01	10	10	0.5	0.5	1	0	0.5	0
c	00	01	10	11	0.5	0.5	0.5	0.5	0	0

Tabla 4.1: Etiquetas binarias resultantes de las tres soluciones representadas en la figura 4.3 y evaluación de las funciones de coste (eq. 4.7) y (eq. 4.3).



(a) (b)

(c)

Figura 4.3: (a) Primera solución . (b) Segunda solución. (c) Tercera solución

binaria anterior igual a 0, $\mu_{m,1}$ es la media binaria de la coordenada m para los N_1 elementos del batch que tengan la coordenada binaria anterior igual a 1.

En el algoritmo 1 se explica el método para optimizar esta función de coste. El algoritmo recorre secuencialmente las dimensiones de los descriptores del batch teniendo en cuenta las coordenadas binarias previas. Si no se tuvieran en cuenta y simplemente se buscara que la media binaria de las coordenadas fuera 0.5, existirían varias posibles soluciones que minimizarían la función. Se muestra lo que ocurriría con un contraejemplo. Supongamos que la salida de la red es un descriptor de 2D y el número de clases es 4. Si no se tiene en cuenta el algoritmo 1 se dan las siguientes situaciones, (ver figura 4.3).

En el caso de encontrarnos ante la situación (a) o (b), la red no separa las clases en los cuadrantes binarios, por consiguiente la binarización de los descriptores resultaría poco precisa. No obstante, si nos encontramos ante la situación (c) la binarización trivial, asignaría una etiqueta binaria única a cada clase. Como podemos ver en la última columna de la tabla 4.1, si no aplicamos el algoritmo 1 propuesto, la función de coste (eq. 4.3) genera un coste $L = 0$ para las tres soluciones, mientras que la función de coste (eq. 4.7) solo genera coste nulo $L_s = 0$ para la solución (c). Es por ello que el

Algorithm 1: Función de distribución homogénea

Data: $\{F(\mathbf{x}_1, W) \dots F(\mathbf{x}_N, W)\}$; batch de descriptores
Result: L_s y L'_s; Coste de la función y su gradiente
for $m \leftarrow 1$ **to** M **do**
  if $m = 1$*; primera coordenada* **then**
    for $n \leftarrow 1$ **to** N **do**
      optimizar_pesos_para_hacer_$\mu_1 = 0{,}5$
  else if $\mu_{m-1} = 0{,}5$ *; la coordenada anterior* **then**
    for $n \leftarrow 1$ **to** N **do**
      if $b_n(m-1) = 0$*; coordenada anterior=0* **then**
        optimizar_pesos_para_hacer_$\mu_{m,0} = 0{,}5$;
      else
        optimizar_pesos_para_hacer_$\mu_{m,1} = 0{,}5$;
  else
    for $n \leftarrow 1$ **to** N **do**
      No_optimizar_pesos;

algoritmo 1 logra maximizar las distancias de Hamming.

4.3. Clasificador basado en descriptores binarios

El entrenamiento de un red siamesa distribuye en clusters separables las diferentes clases, sin embargo, dado que el etiquetado de la base de datos es 1 o 0 se desconoce la clase a la que pertenece cada cluster. Para asignar las clases a los clusters se necesita una base de datos con las etiquetas de cada clase (Ground Truth). Por ello, utilizamos el clasificador *K-nearest-neighbour* para comprobar la precisión de los métodos de binarización propuestos.

***K*-nearest-neighbour:** es un método no paramétrico utilizado en problemas de clasificación, en donde se asignan etiquetas a descriptores. La asignación se hace mediante la votación de los k-vecinos más cercanos, de tal forma que la clase más votada es la asignada al descriptor de test. Generalmente K suele ser un entero positivo y pequeño, en nuestro caso se ha trabajado con $K = 7$. Este método se aplicará para predecir las etiquetas de los descriptores continuos y de los descriptores binarios.

Capítulo 5

Experimentos y resultados

En este capítulo se proponen diferentes arquitecturas para la binarización de descriptores. Para comprobar su desempeño se realizan entrenamientos supervisados con la base datos MNIST. Se ilustrarán los experimentos para 4 clases con descriptores de 2 dimensiones para facilitar la visualización y explicación de los resultados.

5.1. Métricas

Para evaluar el funcionamiento de la binarización disponemos de la base de datos de test del MNIST. Se quiere comprobar que las etiquetas asignadas por el clasificador corresponden con la solución de referencia (Ground Truth). Se utilizarán las siguientes métricas:

Precisión o Fracción de correctos (fc): es la fracción de etiquetas asignadas que coinciden con la etiqueta de referencia respecto del número de etiquetas asignadas. Trabajaremos con la precisión de descriptores continuos y con la precisión de descriptores binarios.

$$fc\ continuo = \frac{N^o\ de\ etiquetas\ acertadas\ con\ descriptores\ continuos}{N^o\ de\ etiquetas\ evaluadas} \times 100 \quad (5.1)$$

$$fc\ binario = \frac{N^o\ de\ etiquetas\ acertadas\ con\ descriptores\ binarios}{N^o\ de\ etiquetas\ evaluadas} \times 100 \quad (5.2)$$

Matriz de confusión : es una matriz $n \times n$, donde n es el número de clases, que nos permite visualizar el desempeño de un entrenamiento supervisado. Las columnas representan las etiquetas de referencia y las filas las etiquetas asignadas por el clasificador. Es muy útil a la hora de ver si dos clases se están confundiendo entre ellas o para comprobar que el clasificador es capaz de distinguir todas las clases.

5.2. Red siamesa

En este experimento se mantiene la arquitectura de la red propuesta en la sección 2.3, donde la única función de coste es la función contrastive (eq. 2.2), ver la figura 5.1.

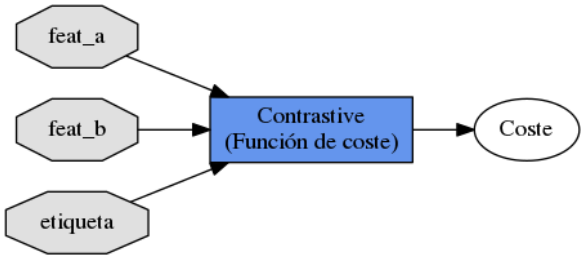
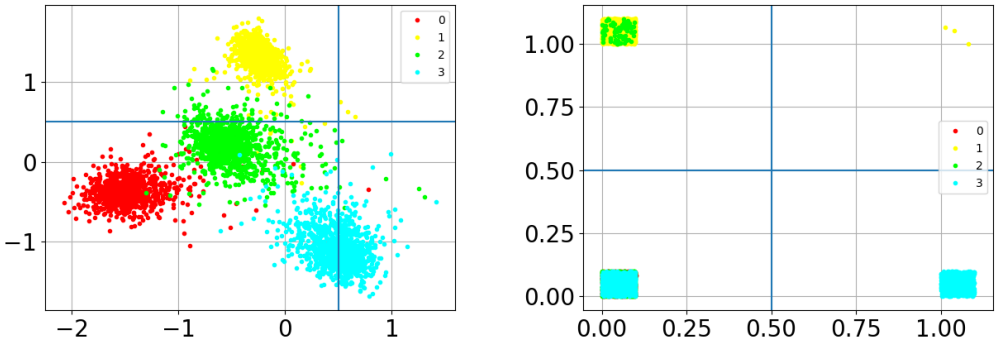


Figura 5.1: Función de coste contrastive. Propuesta, ver figura 2.4.

Como se puede observar en la Figura 5.2a, con la arquitectura propuesta la agrupación en continuo es correcta. Dado que utilizamos la función contrastive los clusters quedan entorno a los valores de inicialización, que son cero. Sin embargo no es correcta la agrupación binaria tras aplicar la binarización trivial (ver Figura 5.2b) ya que no hemos intervenido en la localización de los clusters en el espacio. Para el problema de las diez clases del MNIST los resultados del entrenamiento son los de la Tabla 5.1. En la Figura 5.3 se puede ver como la red confunde las clases binarias.



(a) (b)

Figura 5.2: Con función de coste contrastive, clasificación de cuatro clases. (a) Descriptores continuos. Las líneas azules indican los umbrales de binarización. (b) Clasificación con descriptores binarios. Se añade una perturbación aleatoria uniforme centrada amplitud 0,3 para facilitar su visualización.

Dimensión	Fc continuo	Fc binario
4	97,93 %	58,38 %
8	98,28 %	47,33 %
16	98,72 %	31,33 %

Tabla 5.1: Resultados de test en continuo y binario para la base de datos MNIST.

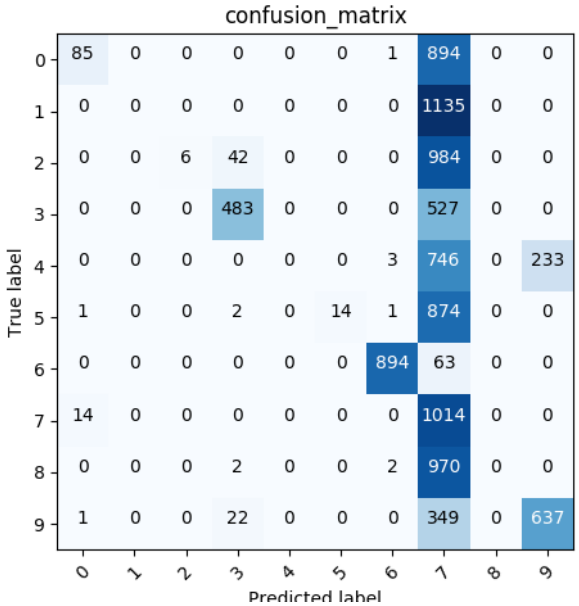


Figura 5.3: Matriz de confusión del entrenamiento MNIST completo con función contrastive y 16 bits.

5.3. Red siamesa con función de activación sigmoide

En este experimento utilizamos la función de activación sigmoide al final de las capas interconectadas para saturar los datos entre cero y uno. Para la agrupación de las clases utilizaremos la función de coste contrastive. La arquitectura de la red se muestra en la figura 5.4.

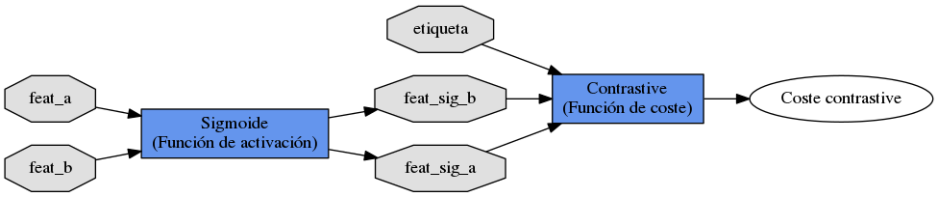
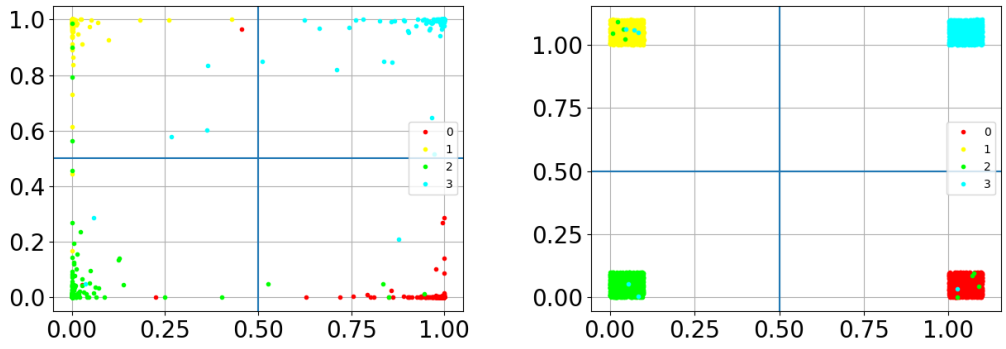


Figura 5.4: Propuesta: Función de coste contrastive y función de activación sigmoide. Propuesta, ver figura 2.4.

Como puede verse en la Figura 5.5a la agrupación en continuo es correcta. Dado que los clusters se encuentran localizados entorno a 0 y 1, y cada clase está en un cuadrante binario obtenemos una clasificación binaria precisa aplicando la binarización trivial a

los descriptores continuos, ver Figura 5.5b. Esta arquitectura de red sí que actúa sobre la localización de los clusters propocionando unos buenos resultados porque como se ve en la Tabla 5.2 la diferencia entre la precisión continua y la binaria es reducida.



(a) (b)

Figura 5.5: Clasificación con función de coste contrastive y sigmoide. (a) Descriptores continuos. Las líneas azules indican los umbrales de binarización. (b) Clasificación con descriptores binarios. Se añade una perturbación aleatoria uniforme centrada amplitud 0,3 para facilitar su visualización.

Hemos utilizado el algoritmo de descenso por gradiente (SGD), que necesita los gradientes calculados mediante *back-propagation*, por ello el uso de la función de activación sigmoide puede ralentizar notablemente el entrenamiento debido al problema del desvanecimiento del gradiente. El gradiente es necesario para que el aprendizaje converja, sin embargo si el gradiente se extingue antes de que el entrenamiento converja, la red no aprende. Cuanto mayores conexiones tenga la red y más capas tenga, más acusado es el efecto del desvanecimiento y dado que el método de binarización se quiere generalizar para más problemas y redes, es necesario buscar alternativas al uso de la función sigmoide.

Dimensión	**Fc continuo**	**Fc binario**
4	89,23 %	84,87 %
8	93,62 %	90,07 %
16	95,35 %	92,08 %

Tabla 5.2: Resultados de test en continuo y binario para la base de datos MNIST.

5.4. Red siamesa con función de coste cuantización

En este experimento utilizaremos la topología propuesta en la sección 2.3 junto con dos funciones de coste; cuantización (eq.4.5) y constrastive (eq.2.2). La contrastive

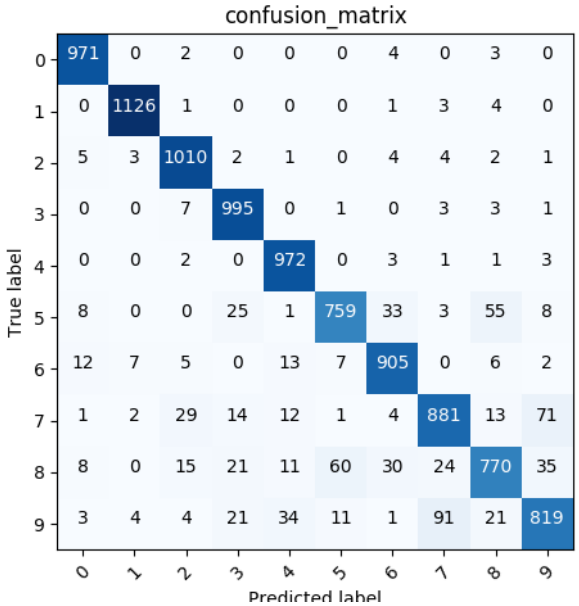


Figura 5.6: Matriz de confusión del entrenamiento MNIST completo con función de activación sigmoide y función de coste contrastive y 16 bits.

clusteriza las clases y la cuantización fuerza a que los clusters tengan sus coordenadas entorno a cero y a uno. La arquitectura de la propuesta se muestra en la figura 5.7.

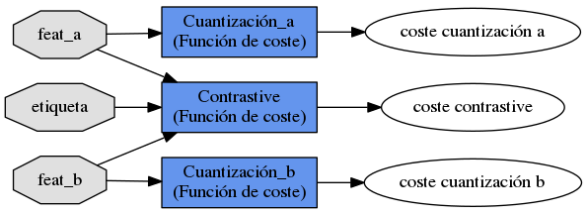
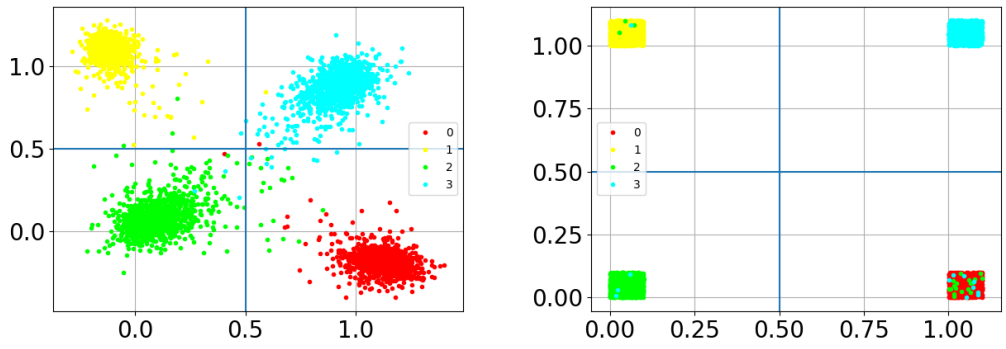


Figura 5.7: Arquitectura propuesta: función contrastive y función de cuantización. Propuesta, ver figura 2.4.

En la Figura 5.8 se puede ver como esta topología ha logrado un entrenamiento en el que los descriptores se agrupan en clusters entorno a las coordenadas cero y uno, proporcionando una buena clasificación binaria aplicando la binarización trivial.

En la Tabla 5.3 y en la Figura 5.9 queda reflejado que la función cuantización (eq.4.5) garantiza que las dimensiones de los descriptores en continuo van a estar entrono a los valores cero y uno, pero no garantiza que la binarización trivial sea precisa. Esto es debido a que esta arquitectura no actúa sobre la ocupación de los clusters en los cuadrantes binarios que maximiza la distancia de Hamming entre las clases. Por ello, la precisión de la binarización sigue teniendo una fuerte dependencia con la inicialización de la red.



(a) (b)

Figura 5.8: Clasificiación de 4 clases con descriptores de 2 dimensiones. Entrenamiento realizado con función contrastive y cuantización. (a) Descriptores continuos. Las líneas azules indican los umbrales de binarización. (b) Clasificación con descriptores binarios. Se añade una perturbación aleatoria uniforme centrada amplitud 0,3 para facilitar su visualización.

Dimensión	**Fc continuo**	**Fc binario**
4	89,1 %	83,81 %
8	92,10 %	67,80 %
16	91,22 %	29,14 %

Tabla 5.3: Resultados en continuo y binario del entrenamiento con función contrastive y cuantización con los datos de test para la base de datos MNIST. Los descriptores son de 4, 8 y 16 dimensiones

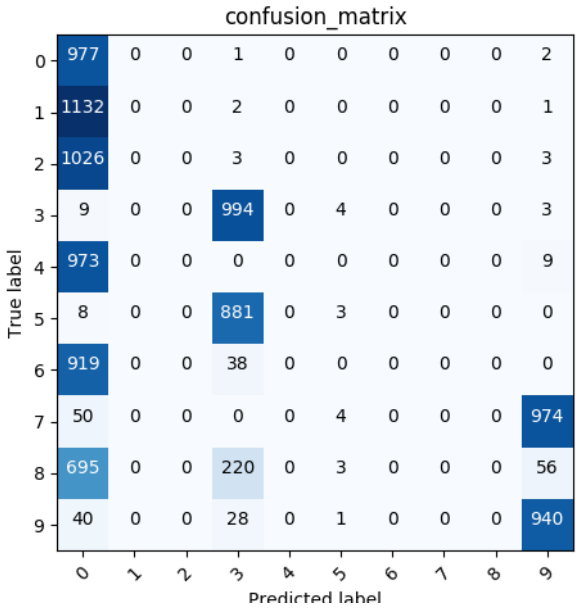


Figura 5.9: Matriz de confusión del entrenamiento MNIST completo con función cuantización y contrastive y 16 bits.

5.5. Deep constrained clusterization and binarization (DCCB)

El DCCB es una arquitectura de red propuesta en este trabajo que logra una mayor restricción en la localización de los clusters en el espacio. Consiste en añadir al final de una red siamesa, en nuestro caso la propuesta en la sección 2.3, una función de activación ReLU, una función de traslación, y las funciones de coste contrastive y cuantización. La arquitectura de la red se muestra en la Figura 5.10.

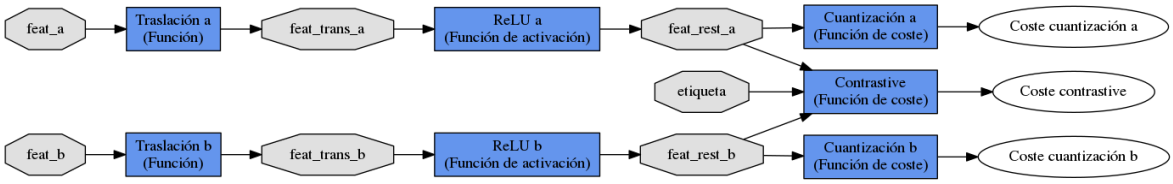


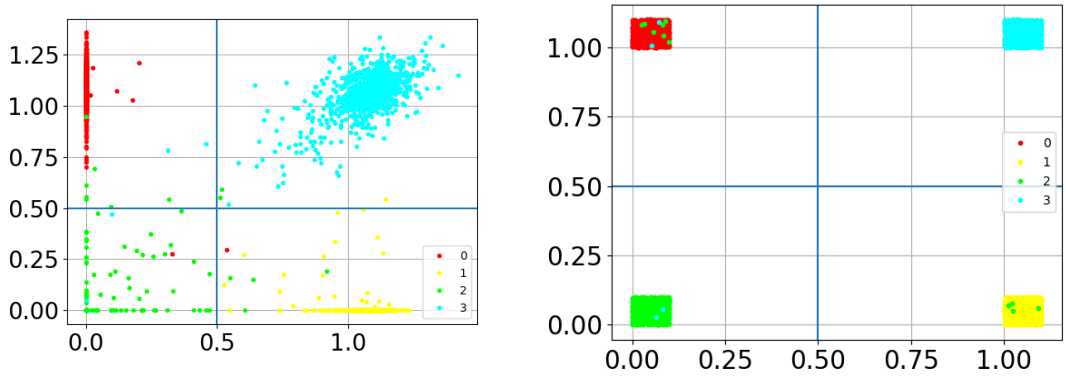
Figura 5.10: Arquitectura propuesta: DCCB. Propuesta, ver figura 2.4.

La función de activación ReLU y la función traslación tienen la finalidad de restringir la localización de los descriptores y reducir el efecto de la aleatoriedad de la inicialización de los pesos. La ReLU satura a cero los valores que sean negativos. Así pues, al colocar esta función de activación forzamos que los valores de los descriptores sean positivos impidiendo que los clusters aparezcan en cuadrantes negativos. Dado que el umbral de binarización es 0.5, con la función traslación desplazamos todos los elementos de los descriptores 0.5 unidades, facilitando que el entrenamiento de la red este centrado entorno a 0.5. En esta situación, se podría atribuir su buen funcionamiento, a que la naturaleza de la función cuantización (ver Figura 4.2) al centrar el entrenamiento en 0.5 aumente la probabilidad de que dos clases diferentes converjan a distintos mínimos y por consiguiente, se maximice la distancia de Hamming entre los descriptores binarios, (ver Figura 5.11).

Dimensión	Fc continuo	Fc binario
4	87,58 %	84,23 %
8	92,43 %	91.42 %
16	93,84 %	91,59 %

Tabla 5.4: Resultados en continuo y binario del entrenamiento con los datos de test para la base de datos MNIST. Los descriptores son de 4, 8 y 16 dimensiones

Como puede verse en la tabla 5.4 y en la matriz de confusión (Figura. 5.12) el DCCB proporciona unos resultados mejores que el apartado anterior (ver Tabla 5.3) en la binarización de descriptores. El DCCB no logra un control total sobre la localización de los clusters en el espacio, ya que el resultado del entrenamiento en menor medida sigue



(a) (b)

Figura 5.11: Clasificiación de 4 clases con descriptores de 2 dimensiones. Entrenamiento realizado con DCCB. (a) Descriptores continuos. Las líneas azules indican los umbrales de binarización. (b) Clasificación con descriptores binarios. Se añade una perturbación aleatoria uniforme centrada amplitud 0,3 para facilitar su visualización.

dependiendo de la inicialización de los pesos. Sin embargo, esta aleatoriedad se termina supliendo aumentando las dimensiones de los descriptores, que como puede verse en la Tabla 5.4 proporciona mejores resultados conforme se aumentan las dimensiones del descriptor. Comparándonos con los diez mejores métodos de clasificación con descriptores binarios nos situamos el puesto número dos del ranking propuesto por [5] con una precisión binaria del 96,6 % con descriptores de 32 bits (ver Tabla 5.5).

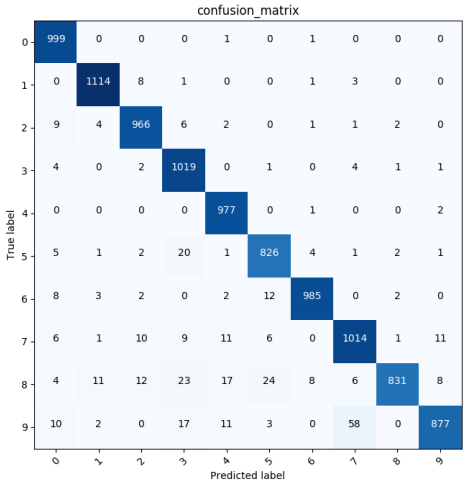


Figura 5.12: Matriz de confusión del entrenamiento MNIST completo con el método DCCB y 16 bits.

Método	# Bits	Fc binario
FastHash[5]	64	0.972
DCCB	**32**	**0.969**
FSDH[6]	64	0.965
SDH[7]	64	0.963
KSH[8]	64	0.927
AGH[9]	64	0.899
IMH[10]	64	0.897
CCA-ITQ[11]	64	0.894
PCA-ITQ[12]	64	0.886
BRE[13]	64	0.839

Tabla 5.5: Comparación con los diez mejores métodos para la clasificación con descriptores binarios del MNIST con entrenamiento supervisado según [5]

5.6. Deep directed clusterización and binarization (DDCB)

El DDCB es una una arquitectura de red propuesta en este trabajo que proporciona un descriptor binario único para cada clase. La arquitectura consiste en una red siemesa con tres funciones de coste: cuantización, distribución homogénea y contrastive. La arquitectura se muestra en la Figura 5.13.

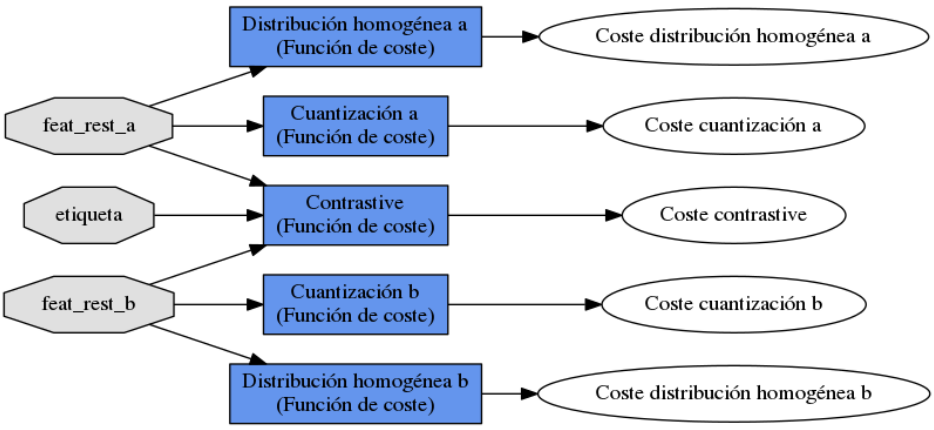


Figura 5.13: Arquitectura propuesta: DDCB.Propuesta, ver figura 2.4.

En este método la función cuantización centra las coordenadas de los descriptores entorno a cero y a uno. La función contrastive juntará aquellos descriptores que pertenezcan a la misma clase y la distribución homogénea separará los descriptores de diferentes clases. Esta técnica de agrupamiento nos permite controlar la dirección de la separación de los descriptores, lo que se traduce en un mayor control de la localización de los clusters en el espacio, independientemente de la inicialización. El entrenamiento de la DDCB permite obtener unos descriptores binarios precisos mediante la aplicación

de la binarización trivial. El método proporciona muy buenos resultados si el número de clases entrenadas coincide con 2^n (ver Tabla 5.6).

Dimensión	clases	Fc continuo	Fc binario
3	8	99,18 %	99,17 %
4	10	93,52 %	50,11 %

Tabla 5.6: Resultados en continuo y binario del entrenamiento con 8 clases del MNIST, para descriptores de 3 dimensiones

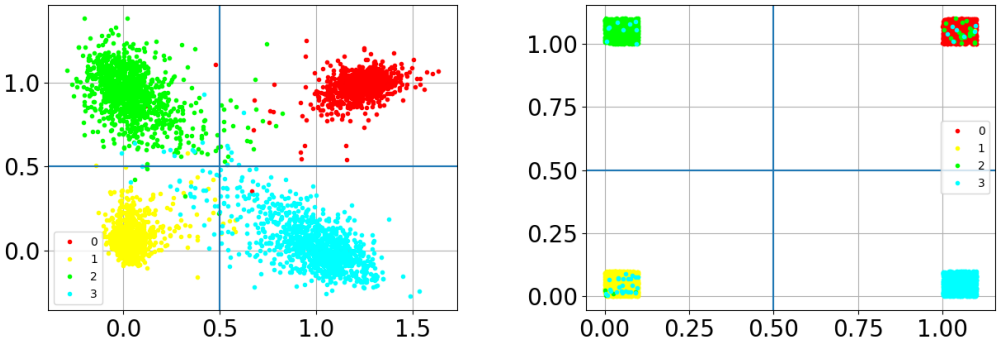


Figura 5.14: Ditribución de los clusters de las 4 clases. Entrenando con el método DDCB. Las rectas azules marcan los umbrales de binarización

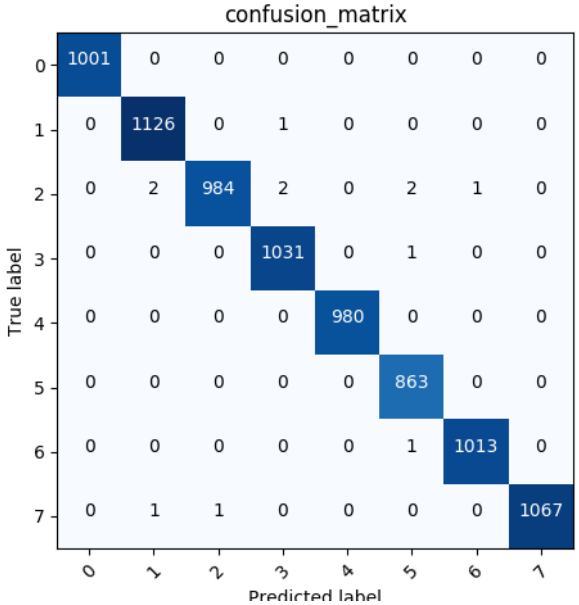


Figura 5.15: Matriz de confusión del entrenamiento con 8 clases del MNIST, DDCB.

Como puede verse en la Figura 5.14 los clusters están centrados en los mínimos globales cero y uno, además de que cada cluster ocupa un único cuadrante binario, obteniendo así una etiqueta binaria única para cada clase. En la matriz de confusión de la

Figura 5.15 y en la Tabla 5.6 se puede apreciar la alta precisión de la DDCB, que no solo clasifica correctamente en binario sino que utiliza el mínimo número de bits necesario para representar 8 clases.

Capítulo 6

Conclusiones y discusión de resultados

Los experimentos realizados muestran que se pueden obtener descriptores binarios a partir de los descriptores continuos generados por una red siamesa, interviniendo en la localización de los clusters durante el entrenamiento.

El método DCCB nos proporciona los resultados más prometedores, colocándonos en el segundo puesto en el estado del arte en la binarización de descriptores para la base de datos MNIST. El objetivo a largo plazo es implementar este método en problemas de reconocimiento de lugares, de rostros, animales u objetos entre otros, en definitiva, dar el salto al mundo real. El DCCB únicamente afecta al final de la arquitectura de la red, lo que supondría una sencilla implementación en redes como las VGGs-16.

Por otro lado, el DDCB presenta la limitación de que pierde eficiencia si el número de clases a entrenar no es 2^n y de que se necesitan conocer el número de clases a entrenar. Sin embargo, es el único método que garantiza una etiqueta binaria única para cada clase y que utiliza el mínimo número de bits posible, con un rendimiento tan alto. Por ello resultaría de interés continuar investigando en su desarrollo.

Capítulo 7

Herramientas

Para la composición de las redes y su entrenamiento se han utilizado las librerías de Caffe [2] por su diversidad y facilidad a la hora de diseñar diferentes topologías. Las funciones creadas en el trabajo se han implementado en C++ para añadirlas como capas a nuestra librería de Caffe. Para la comprobación del funcionamiento y las gráficas se ha utilizado Python, especialmente las librerıas Numpy, SciPy, MatPlotLib y OpenCV.

Capítulo 8

Bibliografía

[1] M. Calonder, V. Lepetit, M. Ozuysal, T. Trzcinski, C. Strecha, and P. Fua. Brief: Computing a local binary descriptor very fast. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 34(7):1281–1298, July 2012.

[2] W. Liu, J. Wang, R. Ji, Y. Jiang, and S. Chang. Supervised hashing with kernels. In *2012 IEEE Conference on Computer Vision and Pattern Recognition*, pages 2074–2081, June 2012.

[3] Raia Hadsell, Sumit Chopra, and Yann LeCun. Dimensionality reduction by learning an invariant mapping. In *2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition, CVPR 2006*, 2006.

[4] Kevin Lin, Jiwen Lu, Chu-Song Chen, and Jie Zhou. Learning compact binary descriptors with unsupervised deep neural networks. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2016.

[5] J. Gui, T. Liu, Z. Sun, D. Tao, and T. Tan. Fast supervised discrete hashing. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 40(2):490–496, Feb 2018.

[6] Guosheng Lin, Chunhua Shen, Qinfeng Shi, Anton van den Hengel, and David Suter. Fast supervised hashing with decision trees for high-dimensional data. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2014.

[7] J. Gui, T. Liu, Z. Sun, D. Tao, and T. Tan. Fast supervised discrete hashing. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 40(2):490–496, Feb 2018.

[8] W. Liu, J. Wang, R. Ji, Y. Jiang, and S. Chang. Supervised hashing with kernels. In *2012 IEEE Conference on Computer Vision and Pattern Recognition*, pages 2074–2081, June 2012.

[9] Wei Liu, Jun Wang, Sanjiv Kumar, and Shih-Fu Chang. Hashing with graphs. In *Proceedings of the 28th International Conference on International Conference on Machine Learning*, ICML'11, pages 1–8, USA, 2011. Omnipress.

[10] Fumin Shen, Chunhua Shen, Qinfeng Shi, Anton van den Hengel, and Zhenmin Tang. Inductive hashing on manifolds. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2013.

[11] Y. Gong, S. Lazebnik, A. Gordo, and F. Perronnin. Iterative quantization: A procrustean approach to learning binary codes for large-scale image retrieval. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 35(12):2916–2929, Dec 2013.

[12] Y. Gong, S. Lazebnik, A. Gordo, and F. Perronnin. Iterative quantization: A procrustean approach to learning binary codes for large-scale image retrieval. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 35(12):2916–2929, Dec 2013.

[13] Brian Kulis and Trevor Darrell. Learning to hash with binary reconstructive embeddings. In Y. Bengio, D. Schuurmans, J. D. Lafferty, C. K. I. Williams, and A. Culotta, editors, *Advances in Neural Information Processing Systems 22*, pages 1042–1050. Curran Associates, Inc., 2009.

[14] Yangqing Jia, Evan Shelhamer, Jeff Donahue, Sergey Karayev, Jonathan Long, Ross Girshick, Sergio Guadarrama, and Trevor Darrell. Caffe: Convolutional architecture for fast feature embedding. In *Proceedings of the 22Nd ACM International Conference on Multimedia*, MM '14, pages 675–678, New York, NY, USA, 2014. ACM.

Lista de Figuras

Lista de Tablas